第2章 项目2——五子棋游戏

一、项目概述

通过本项目的实践，了解系统总体分析的方法，能够分析系统的框架图和总体流程图；能够分析游戏的框架图和总体流程图，并根据系统总体流程，抽取出所需要的类，确定类之间的调用关系；能够完成系统设计，并按照系统设计要求进行系统测试。项目完成后，学生应该具备开发小型的应用系统的能力。

该实践培养学生进一步深入理解面向对象的程序设计思想；培养学生具有分析问题和解决问题的能力；初步培养学生具有一定的自学能力和代码编写能力。使学生养成遵循工程规范的习惯和专业素养，项目采用团队开发模式，培养学生的团队协作能力。

二、知识要点及掌握程度

(1)需求获取方法和需求文档的撰写方法：运用。
(2)用户界面设计方法：运用。
(3)类的抽取和设计方法：组织。
(4)人工智能算法：了解。
(5)软件设计文档的撰写方法：运用。
(6)软件实现流程：设计。
(7)软件测试的方法和过程：了解。

三、能力要点及掌握程度

(1)系统的显现和交互作用：识别系统单元间的接口，确定功能模块。
(2)分析问题：掌握软件工程系统分析的方法，能够通过分析获取系统需求。
(3)具有概念化和抽象化能力：能够利用面向对象思想对现实问题进行抽象化。
(4)引进、消化、吸收再创新能力：培养学生的创新能力。
(5)查阅印刷资料和电子文献：应用图书馆工具(在线检索、数据库、搜索引擎等)检索并获取信息。
(6)行业的基本规范：熟悉 C＃语言编程基本规范，养成遵循工程规范的习惯和专业素养。
(7)行业应用技术：熟悉 Visual Studio 2010 开发平台，掌握数据库等应用开发技术。
(8)软件实现过程：能够使用 C＃语言编程实现系统功能。
(9)测试、证实、验证及认证：掌握简单的测试方法，统计各阶段的代码量等指标。

2.1　游戏构思

作为游戏项目，软件开发者要考虑的因素很多，不仅限于项目的技术实现（游戏引擎算法等），从某种意义上讲，游戏项目对游戏的界面（是否漂亮、是否符合玩家习惯等）、游戏的可玩性（游戏情景、脚本等）、游戏体验（角色成长、响应速度等）的要求会更高些。所以本章我们要实现的项目——五子棋游戏，在项目构思上，只有在完美体现五子棋游戏通用玩法的同时，更好的考虑游戏项目的特征，才能设计出真正玩家想要的五子棋小游戏。

2.1.1　项目背景

五子棋游戏是一款经典的棋类游戏，有着悠久的历史和众多的玩家。随着人们生活水平的提高，计算机走进寻常百姓家。棋类游戏软件成为深受人们喜爱的计算机游戏软件，为人们的休闲娱乐和智商培养提供了平台。五子棋游戏只是众多游戏中的一种，而且又可分为各种版本，主要是单机版和网络版，但这里以讨论单机版五子棋游戏为主，网络版可留作后续提高开发。

五子棋游戏具有棋类游戏的共性特点，同时实现相对简单，作为学生实践学期的训练项目开发比较适合。不过不要小看五子棋游戏，把它看作是小孩子玩儿的游戏，它的另一个正式名字是“连珠”，现在还有世锦赛呢！

现简述其规则，有一个 15×15 的棋盘，棋手分黑白两方，黑方持黑，白方持白，交替落子。一方棋子首先形成 5 子相连则该方胜（相连方向包括正东正西、正南正北、西北东南、东北西南四个方向）。

因为五子棋先手（先落子一方）有巨大优势，理论上讲，对于有一定棋力的棋手都能够保证先行必胜，为了平衡先手优势，定义了禁手规则。设定黑方先行，即给黑棋增加禁手限制。

关于禁手的几个问题：

（1）白棋无禁手，黑棋有禁手，且禁手判负。

（2）白棋不仅自己没有禁手，还可以运用禁手规则来制定战术，通过某种手段来逼迫黑棋禁手，这是白棋的基本战术之一。称为“追下取胜”，俗称“捉禁手”。

（3）那么白棋岂不是很讨便宜，事实是，加上了禁手规则之后，大多数开局仍然是黑优。没有错，先走一步，仅仅这一步，优势就这么大。

关于禁手的种类：

（1）三三禁手：黑方一步落下同时形成两个活三，此步为三三禁手。

注意：这里一定要两个都是活三才算数。

（2）四四禁手：黑方一步落下同时形成两个四，此步为四四禁手。

注意：只要是两个“四”即为禁手，无论是那种四。活四，跳四，冲四都算数。

（3）四三三禁手：黑方一步使一个四，两个活三同时形成。

（4）四四三禁手：黑方一步使两个四，一个活三同时形成。

（5）长连禁手：黑方超过 5 个子连在一起。

为了简化问题，这里只考虑(1)(2)(5)的禁手情况(很显然(3)的禁手同时也形成(1)的禁手，(4)的禁手同时也形成(2)的禁手)。关于禁手的图解，如图 2-1 所示。

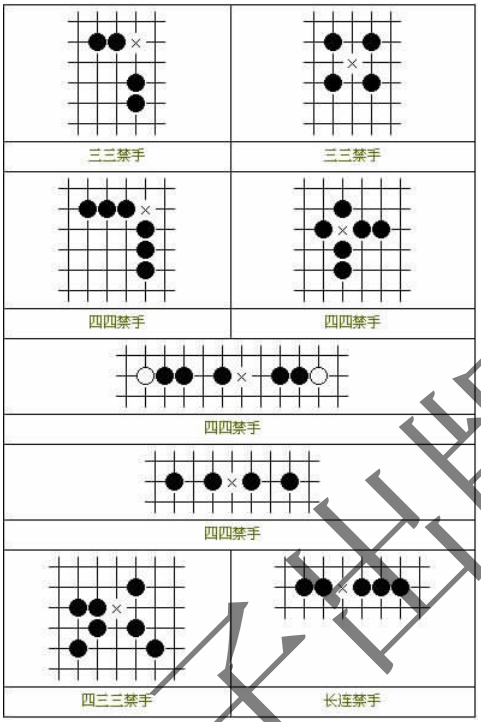

图 2-1 禁手示意图

在这么多禁手的情况下，黑棋如何才能取胜呢？黑棋只有四三胜才可以，即冲四活三胜。当然，活三对方没看见也可以，那称之为自由胜。图 2-2 就是冲四活三胜，即用方框标记的第 33 手。这步同时形成了一个斜着的活三和一个横着的冲四。

在明确了五子棋游戏的背景和规则之后，就可以动手做我们有趣的五子棋游戏啦！

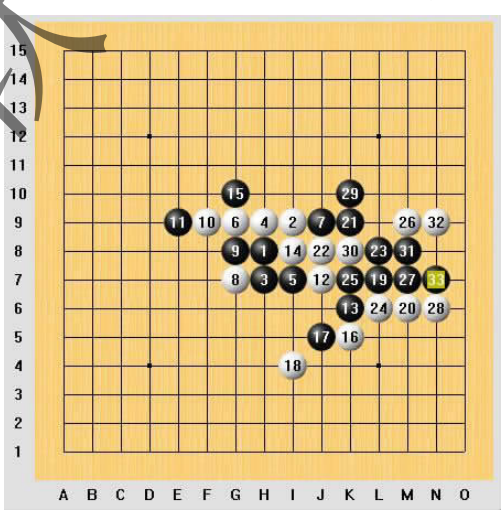

图 2-2 冲四活三胜形状举例

2.1.2　项目需求

1. 功能需求

(1)棋盘、棋子的绘制。

此案例是一个 GUI 的程序,更形象、直观,对于棋盘、棋子的绘制可以采用 GDI+的 API 绘画,也可以使用 System.Drawing 命名空间里的 Image 类来完成。从视觉效果上看,后者更有优势。

(2)人人对弈,人机对弈。

所谓人人对弈,是指对弈的是两个人,计算机程序就相当于是一个电子化的提供棋盘、棋子的工具,再加上能够判断落子合法性,棋局结束与否的辅助功能;而人机对弈,是指对弈的双方,一方是人,另一方是计算机程序,要求计算机程序能够模拟人的方式决定落子点,当然计算机程序的落子点选择也是尽量优化,以取胜为目的的,这个计算机走棋的游戏引擎可以有多种算法来实现,定义的启发式规则也有多种,从而体现在计算机程序的棋力上。

(3)历史记录信息显示。

对于黑、白双方的走棋过程进行记录,可以方便悔棋或重现对弈的过程。

(4)落子点信息、玩家信息、胜负和错误信息的提示。

在游戏过程中,玩家的误操作常会发生,在走法不满足规则的时候要能够给予相应提示。在胜负可定的情况下,程序应能够判别并给出提示。落子点的信息提示和玩家信息提示是为了方便用户实时了解棋面的简单而使用的辅助功能。

(5)控制信息的设置,如先后手、难度、悔棋等。

五子棋的先手和后手对玩家胜负影响重大,理论上讲如果没有任何限定,对于有一定棋力的棋手而言都能够做到先行必胜,所以要求程序能够设定先后手。在人机对弈的过程中,计算机程序棋力的强弱,能够提升五子棋游戏的娱乐性,所以程序应该能够根据玩家的设定改变计算机程序的棋力,在后台实现上体现为对走棋算法的改变。悔棋,一个不雅的行为,不建议大家干这样的事,但是还是提供这个功能比较好,人非圣贤,谁能无过。

(6)玩家战绩记录及排名。

为玩家建立游戏记录档案,记录其胜率,排名可按胜率排名。胜率是取胜的次数除以玩的总次数。但有一点要注意,如果程序有难度设定功能,那么对于同一玩家要分别记录在不能棋力难度上的胜率,这才公平。排名统计时,也可在不同难度级别上分别进行。

(7)可以修改棋盘大小。

一般情况下,五子棋棋盘的大小设定为 15×15 的,也可以考虑是否能通过设置自定义棋盘的大小呢?这种棋盘大小的改变对五子棋游戏有影响吗?

2. 非功能需求

(1)界面友好,逼真。

(2)下棋落子要有模拟声音,既形象又可提示玩家。

(3)有较详细的游戏玩法帮助。

上面列出的需求并非尽善尽美,大家如果发挥自己的才智,结合用户的需求,还可以对游戏要实现的需求进行更有益、有挑战的扩展。更多需求(有部分与上面重复)如图 2-3 所示。

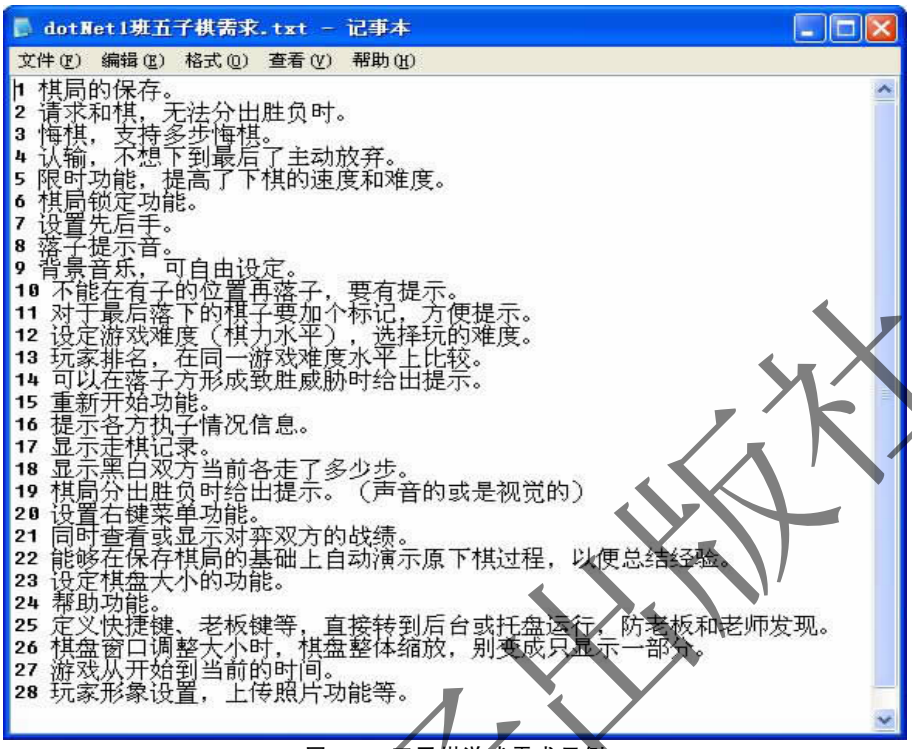

图 2-3 五子棋游戏需求示例

根据用户的功能需求，绘制用例图，如图 2-4 所示。

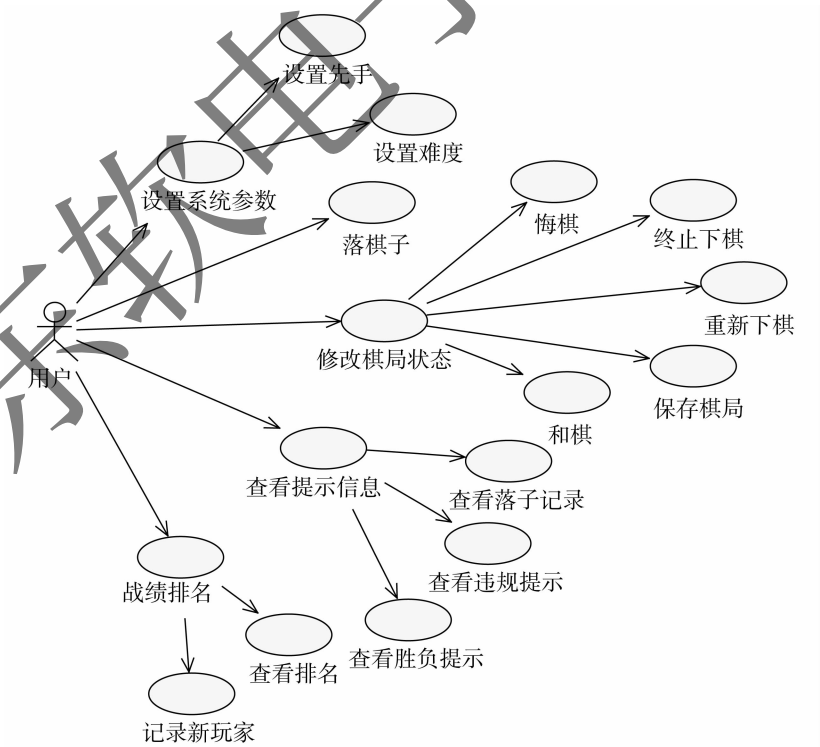

图 2-4 五子棋游戏用例图

接下来为每个用例撰写用例文档，这里以用户落棋子为例，给出用例文档，如表 2-1 所示。请大家完成其他用例的用例文档。

表 2-1　　落棋子用例

用例编号	uc001
用例名称	落棋子
参 与 者	用户(玩家)
用例说明	用例起始于用户使用鼠标在棋盘上点击。
前置条件	新游戏已开始
后置条件	无
基本流程	
参与者的动作	系统动作
1.用户使用鼠标在棋盘格交叉点处点击。	2.系统在指定的位置上绘制棋子。
分支流程	
1.a 用户使用鼠标在棋盘格交叉点外点击。	2.a 系统无响应。
1.b 用户使用鼠标在已落子位置点击。	2.b 系统提示“此处不能落子”。

在明确了五子棋游戏的背景和需求之后，我们就可以动手开发有趣的五子棋游戏啦！

2.2　游戏设计

2.2.1　基本设计

1. 系统功能模块分析

五子棋游戏的体系结构，如图 2-5 所示。

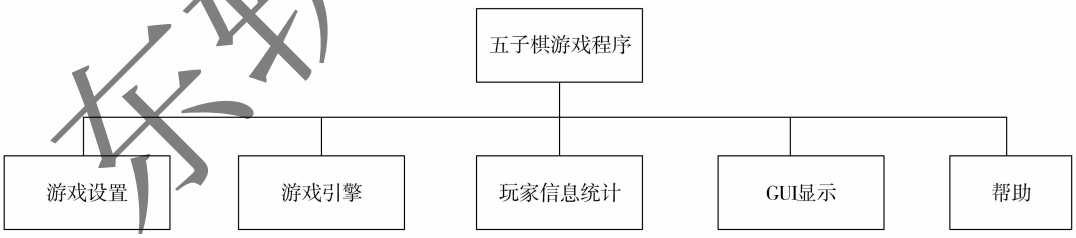

图 2-5　五子棋游戏程序体系结构

2. 界面设计

下面给出程序界面的参考原型示例，如图 2-6～图 2-9 所示，在具体设计中可以根据需要完善。

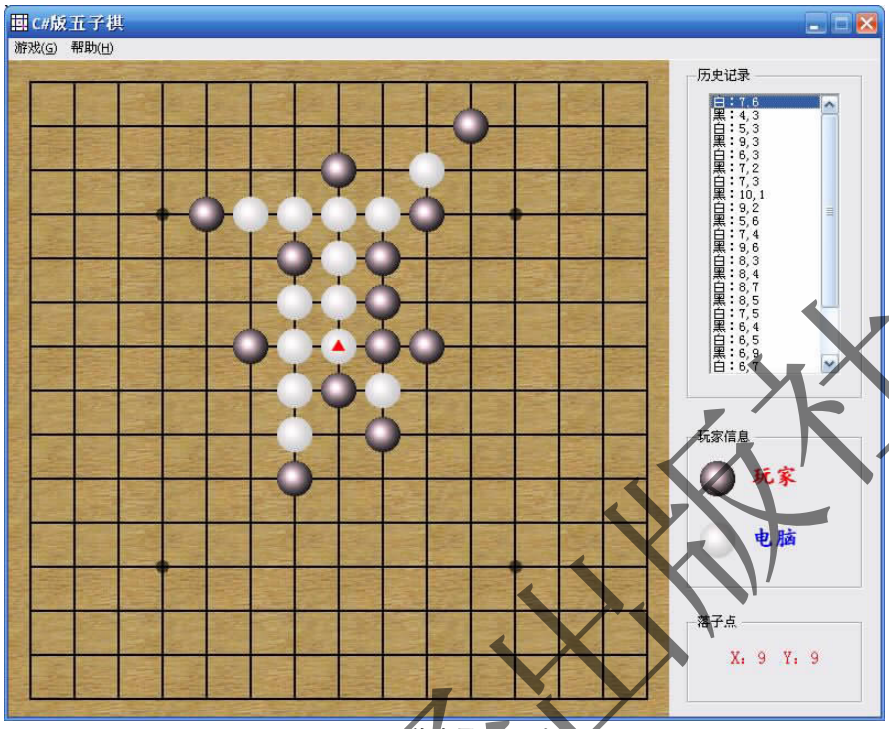

图 2-6 游戏界面原型 1

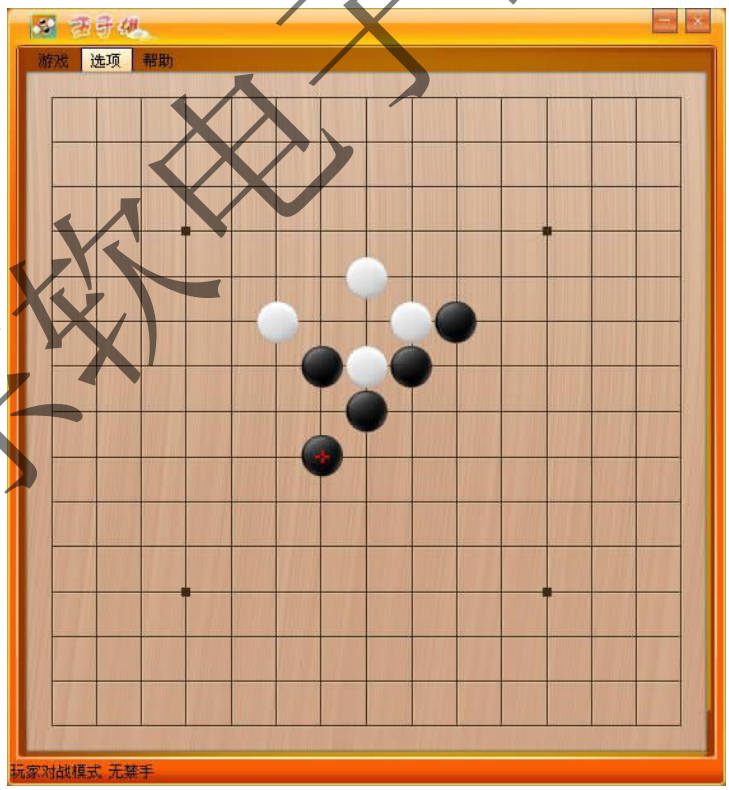

图 2-7 游戏界面原型 2(精简版)

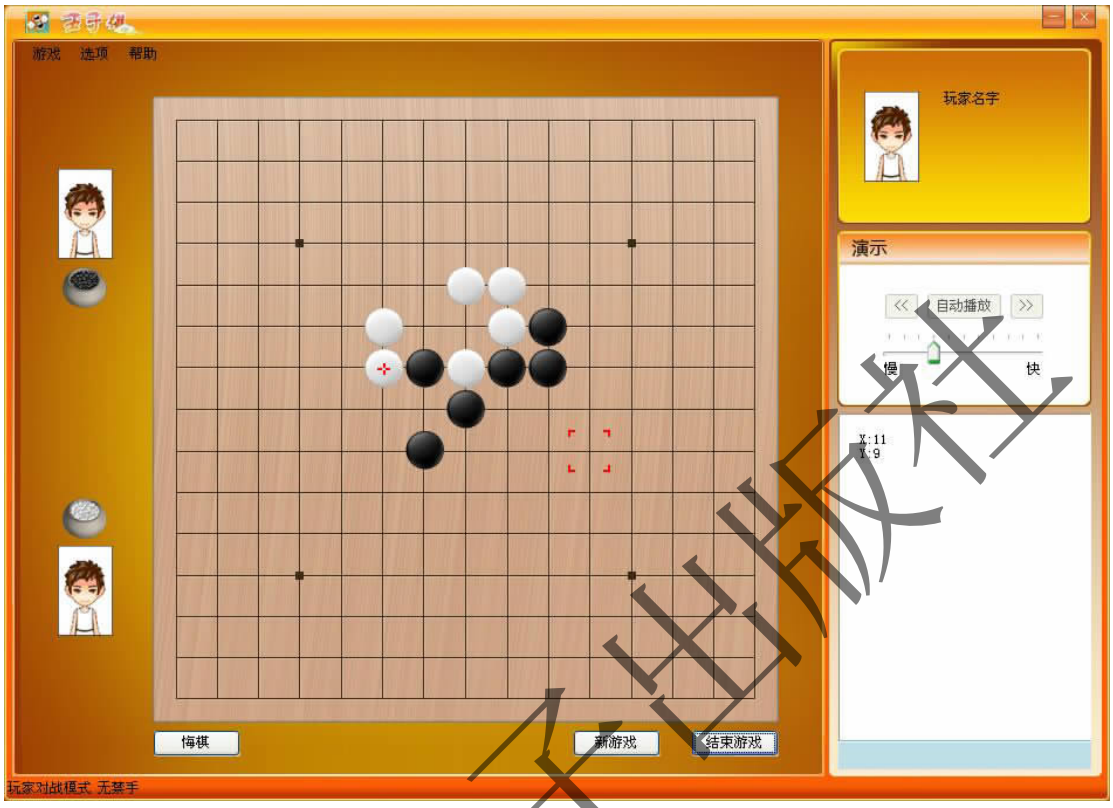

图 2-8　游戏界面原型 2(完整版)

图 2-9　游戏界面原型 2(菜单示例)

3. 数据库设计

(1)数据结构设计。

游戏过程中对于走棋历史记录的保存使用堆栈结构。对于堆栈结构的实现很多平台都有成熟的类实现,不必自行设计。

(2)数据库设计。

五子棋游戏不同于 MIS,需要保存的数据信息比较少,设计了一个玩家记录表。用来记录玩家的信息。将来用于玩家胜率统计和成绩排名。

a. 数据库的概念设计,如图 2-10 所示。

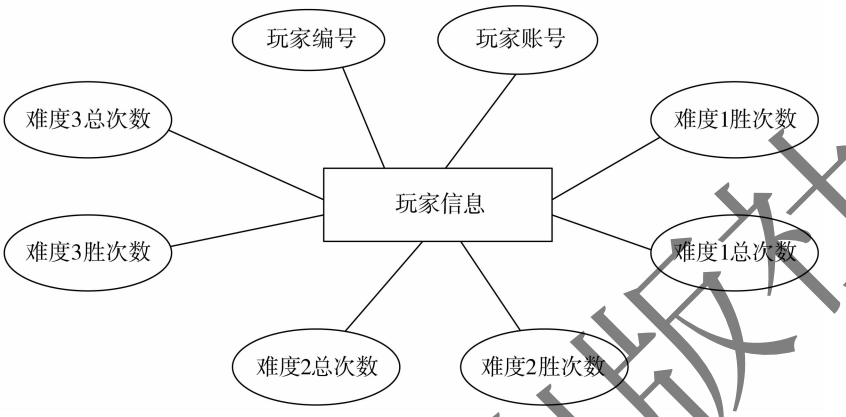

图 2-10 玩家信息实体 ER 图

b. 数据库的逻辑设计,如表 2-2 所示。

表 2-2 玩家信息表

字段名	数据类型	主键	可以为空值	描述
P_ID	varchar	是	否	玩家编号
P_Name	varchar	否	否	玩家账号
P_Hard1Win	int	否	是	难度 1 胜次数
P_Hard1Total	int	否	是	难度 1 总次数
P_Hard2Win	int	否	是	难度 2 胜次数
P_Hard2Total	int	否	是	难度 2 总次数
P_Hard3Win	int	否	是	难度 3 胜次数
P_Hard3Total	int	否	是	难度 3 总次数

4. 类设计

本案例采用面向对象的开发思想进行开发,面向对象技术自从其产生开始,便因其更符合开发规律,而得到广泛的应用。其核心思想是从待解决的问题域中抽象出来,再由类来定义对象,对象是数据和方法的结合体,体现了事物的属性和行为,编程即是按照一定的事务逻辑组织对象进行交互。面向对象的编程,更利于程序的模块化和复用。根据五子棋游戏人机对弈的特点,可以抽象出四个实体类,分别是棋盘类、棋子类、规则类和计算机类。以下简要介绍每个类中的功能,关键属性和方法。

(1)棋盘类(Chessboard)。

棋盘类的属性包括描述棋盘当前状态的二维整型数组,这个数组中的元素值能够表达每个棋盘位置的落子情况,包括该位置没有子、有黑子、有白子三种状态。棋盘类中还要完成对棋盘的绘制,因此还要一个 Graphics 成员。在棋盘上要想根据当前棋局来绘制棋子,显然应该调用棋子对象自身的绘制方法,所以棋子对象也是棋盘类的成员之一。在人机对弈的过程中,在人

走棋后，计算机就要走棋，计算机的走棋也要根据棋盘的状态，所以计算机对象也是棋盘类的成员。对于走棋的过程，棋盘类还应提供一个堆栈结构对象来存储。除此以外，棋盘类中还应该有记录当前棋子是黑还是白，先手玩家是谁的标志变量。

棋盘类的主要方法有根据当前棋盘状态绘制棋盘和棋子的方法；向指定位置落子的方法（落子的过程会涉及到诸多的判断和状态的改变）；人落子的方法和计算机落子的方法，这两种方法主要是给定或是计算出相应落子位置，然后调用前面的向指定位置落子的方法；当然还应该包括棋盘类的初始化方法等。

棋盘类的类图描述，如表 2-3 和表 2-4 所示。

表 2-3　　棋盘（ChessBoard）类字段和属性

字段	属性	描述
m_arrchessboard	Arrchessboard	棋盘情况数组
m_imageGraphics	ImageGraphics	绘制的对象
m_stone	Stone	棋子对象
m_computer	Computer	电脑对象
m_stoneflag	Stoneflag	判断当前棋子是黑（true）是白（false）
m_playfirstflag	playfirstFlage	判断先手玩家是电脑（true）还是人（false），先手下黑棋
m_StarckHistory	StarkHistory	历史记录堆栈

表 2-4　　棋盘（ChessBoard）类方法

方法名称	方法的作用	方法的参数	方法的返回值
Draw	绘制棋盘	无	无
PersonDownStone	计算人下棋的棋子坐标	int，在屏幕上点击的 X 坐标 int，在屏幕上点击的 Y 坐标	无
ComputerDownStone	计算电脑下棋的棋子坐标	无	无
Start	判断是否开始下棋	bool，判断先手玩家是电脑（true）还是人（false）	无

（2）棋子类（Stone）。

同棋盘类一样，棋子类也应该有自身绘制的功能，所以应该有一个 Graphics 成员（与棋盘类 Graphics 成员相同，都是在窗体上绘制，由传递参数得到）和一个绘制自身的方法。如采用调用已有棋子图片的方式来绘制棋子，则可以将存放棋子图片资源的 Image 对象也作为棋子类成员。当然棋子类也要有相应的构造方法。

棋子类的类图描述，如表 2-5 和表 2-6 所示。

表 2-5　　棋子（Stone）类字段和属性

字段	属性	描述
m_imgBlackStone	ImgBlackStone	黑子图片
m_imgWhiteStone	ImgWhiteStone	黑子图片
m_imgRedTriangle	ImgRedTriangle	用于标识最后落子
m_imageGraphics	ImageGraphics	绘制的对象

表 2-6 棋子(Stone)类方法

方法名称	方法的作用	方法的参数	方法的返回值
DrawStone	在正确的地方绘制棋子	int ,在屏幕上点击的 x 坐标 int ,在屏幕上点击的 y 坐标 bool,判断棋子颜色	无

(3)规则类(Rule)。

规则类描述了五子棋游戏的游戏规则,程序中主要是调用它的方法来验证落子方法的可行性和查看棋局的结果等。所以,它不需要含有属性成员,而主要包括一些用来进行判断的方法。如,判断某位置是否可落子的方法;判断在某位置落子后棋局发展结果的方法(失败、胜利、平局、可继续),该方法又调用了判断失败,胜利,平局的方法;在判断失败(对于黑方有禁手失败)、胜利(有五子相连)或是评价某次落子优劣时都需要考虑落子后该子和周围同色棋子构成的形状,分成正东正西、正南正北、西北东南、西南东北四个方向,所以还要有四个方法来检测各方向上的棋子连接和是否可再落子的情况。

规则类的类图描述,如表 2-7 所示。

表 2-7 规则(Rule)类方法

方法名称	方法的作用	方法的参数	方法的返回值
IsExist	判断下棋位置是否有子	int ,在屏幕上点击的 x 坐标 int ,在屏幕上点击的 y 坐标 bool,判断棋子颜色	无
IsFail	检查是否失败, 判断违反了哪些规则	int[,],描述棋盘情况的数组	bool,结果
IsWin	判断是否胜利	int[,],描述棋盘情况的数组	bool,结果
IsTie	判断是否平局	int[,],描述棋盘情况的数组	bool,结果

(4)计算机类(Computer。)

计算机类的核心功能就是能根据当前棋盘局势找到自己最有利的落子点,记录该点坐标,在此基础上就能实现人机对弈了。计算机类的属性需要包括记录要落子点位置横纵坐标的 X、Y 和区别计算机是黑方还是白方的标识成员即可。计算机类的核心方法是找到当前棋盘上最有利于自己落子的位置,具体的算法思想有很多,这里算法的好坏也决定了计算机程序的棋力强弱。本章案例选用一种简单有效,基于落子点权值大小的游戏引擎。在计算出每个可落子点的权值之后,很自然想到还需引入一个从所有权值中找出最大的一个的辅助方法。

计算机类的类图描述,如表 2-8 和表 2-9 所示。

表 2-8 计算机(Computer)类字段和属性

字段	属性	描述
m_ flag	flag	黑子图片
m_x	X	电脑计算出的 X 坐标
m_y	Y	电脑计算出的 Y 坐标

表 2-9　　计算机(Computer)类方法

方法名称	方法的作用	方法的参数	方法的返回值
Down	电脑下棋位置	int[,],描述棋盘情况的数组	无
Check	计算棋盘上每个点的权值	int,棋盘上的横坐标 int,棋盘上的纵坐标 int[,],描述棋盘情况的数组	int,该点对应的权值
MaxQZ	计算权值最大点	int [,],存放权值的数组	void

除了以上提到的 4 个实体类以外,另外一个重要但不需要额外建立的类就是负责和玩家交互,并显示相关信息的窗体类。在窗体类中,我们需要添加各种需要的控件(窗体类的属性成员),并响应事件(事件会调用相关的消息处理函数,即窗体类方法)。本案例中涉及到的事件包括窗体加载、窗体绘制、鼠标移动、鼠标按下等。

2.2.2　高级设计——人工智能设计

1. 人工智能介绍

人工智能(Artificial Intelligence),英文缩写为 AI,它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学,是由人工建立的硬件或软件系统的智能,是无生命系统的智能。例如,一个机器人的智能,一个能与人下棋的软件的智能。智能的表现形式非常广泛,例如,能利用各种知觉系统接受来自环境的各种信息,能形象思维,能逻辑推理,能理解事物之间的内在联系,能作归纳和推广来发现规律;能使自己适应环境,能主动学习,能根据以往经验改进提高自己,不至于犯同样的错误;能进行创新思维来从事各种创新等。智能是一个抽象的概念,一个软件或硬件系统是否有智能,只能根据它所表现出来的行为是否和人类上述的某些行为相类似来做判断。

人工智能在电子计算机发明后不久即创立,是研究人类智能活动的规律,构造具有一定智能的人工系统,研究如何让计算机去完成以往需要人的智力才能胜任的工作,其目的就是要模拟人类的智力活动机制来改进计算机的软件硬件结构,使它们掌握一种或多种人的智能,以便在各种领域内有效替代人的脑力劳动,特别是解决用传统软硬件方法难以解决的问题,如自然语言的翻译、定理证明、模式识别、复杂机器人的控制等。

人工智能领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等。人工智能是一门极富挑战性的科学,将涉及到计算机科学、心理学、哲学和语言学等学科。可以说几乎是自然科学和社会科学的所有学科,其范围已远远超出了计算机科学的范畴。

20 世纪 70 年代以来,人工智能被称为世界三大尖端技术(空间技术、能源技术、人工智能)之一,也被认为是 21 世纪三大尖端技术(基因工程、纳米科学、人工智能)之一。近三十年来它获得了迅速的发展,在很多学科领域都获得了广泛应用,并取得了丰硕的成果,人工智能已逐步成为一个独立的分支,无论在理论和实践上都已自成一个系统。人工智能目前分支比较多,特别是有许多的智能算法,如遗传算法和蚁群算法等的产生和发展,使得原来与人工智能不是很相关的学科也在积极引入人工智能来解决目前面临的困境。下面介绍几种研究和使用非常广泛的人工智能技术。

人工神经网络是人工智能研究范围中产生比较早的一个分支,也是研究比较深入的一个。

顾名思义，人工神经网络就是用计算机来模拟人的神经系统，模仿人类神经（大脑）的工作原理来达到具有智能的目的，人工神经网络是一种应用类似于大脑神经突触连接的结构进行信息处理的数学模型。人脑的神经系统基本单元就是神经元细胞，人脑中神经元大概有 10^{10} 个，每个神经元可能会与其他上千个的神经元相联系，用计算机来模拟这种联系，目前的硬件技术水平还达不到，但是可以简化这种模型来适应当前的硬件技术。目前已经有一些成型的人工神经网络模型可以使用，能到达到使用的目的。神经网络是一种运算模型，由大量的节点（或称“神经元”或“单元”）和之间相互连接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个通过该连接信号的加权值，称之为权重（weight），这相当于人工神经网络的记忆。网络的输出则根据网络的连接方式，权重值和激励函数的不同而不通。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。它的构筑理念是受到生物（人或其他动物）神经网络功能的运作启发而产生的。人工神经网络通常是通过一个基于数学统计学类型的学习方法（Learning Method）得以优化，所以人工神经网络也是数学统计学方法的一种实际应用，通过统计学的标准数学方法我们能够得到大量的可以用函数来表达的局部结构空间，另一方面在人工智能学的人工感知领域，我们通过数学统计学的应用可以来做人工感知方面的决定问题（也就是说通过统计学的方法，人工神经网络能够类似人一样具有简单的决定能力和简单的判断能力），这种方法比起正式的逻辑学推理演算更具有优势。人类的认知过程对外表现即为学习的过程，对内的表现为神经网络的建立及使用过程。

人工神经网络具有以下几个基本特征：

（1）非线性：大脑的智慧就是一种非线性现象，非线性关系是自然界的普遍特性。人工神经元处于激活或抑制两种不同的状态，这种行为在数学上表现为一种非线性关系。具有阈值的神经元构成的网络具有更好的性能，可以极大的提高容错性和存储容量，也方便计算机进行处理。

（2）非局限性：一个可以工作的神经网络通常由多个神经元广泛连接而成。一个系统的整体行为不仅取决于单个神经元的特征，而且可能主要由单元之间的相互作用、相互连接所决定，我们通过各个神经元之间的大量连接来模拟大脑的非局限性。

（3）非常定性：人工神经网络系统具有自适应、自组织、自学习能力。神经网络不但处理的信息可以有各种变化，而且在处理信息的同时，非线性动力系统本身也在不断变化。

人工神经网络中神经元处理单元可表示不同的对象或符号，例如字母、特征、概念，或者一些有意义的抽象模式。网络中处理单元的类型分为三类：输入单元、输出单元和隐藏单元。输入单元接受外部世界的信号与数据；输出单元实现系统处理结果的输出；隐藏单元是处在输入和输出单元之间，不能由系统外部观察的单元。神经元间的连接权值反映了单元间的连接强度，信息的表示和处理体现在网络处理单元的连接关系中。人工神经网络是一种非程序化、适应性、大脑风格的信息处理，其本质是通过网络的变换和动力学行为得到一种并行分布式的信息处理功能，并在不同程度和层次上模仿人脑神经系统的信息处理功能。它是涉及神经科学、思维科学、人工智能、计算机科学等多个领域的交叉学科。人工神经网络是并行分布式系统，采用了与传统人工智能和信息处理技术完全不同的机理，克服了传统的基于逻辑符号的人工智能在处理直觉、非结构化信息方面的缺陷，具有自适应、自组织和实时学习的特点。

当前计算机所要解决的问题往往由于计算能力的限制而不能找到全局最优解，遗传算法是

一种非常优秀的全局寻找最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群(population)开始的,而一个种群则由经过基因(gene)编码的一定数目的个体(individual)组成。每个个体实际上是染色体(chromosome)带有特征的实体。染色体作为遗传物质的主要载体,即多个基因的集合,其内部表现(即基因型)是某种基因组合,它决定了个体形状的外部表现,如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此,在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂,我们往往进行简化,如二进制编码,初代种群产生之后,按照适者生存和优胜劣汰的原理,逐代(generation)演化产生出越来越好的近似解,在每一代,根据问题域中个体的适应度(fitness)大小选择(selection)个体,并借助于自然遗传学的遗传算子(genetic operators)进行组合交叉(crossover)和变异(mutation),产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境,末代种群中的最优个体经过解码(decoding),可以作为问题近似最优解。

遗传算法是解决搜索问题的一种通用算法,对于各种通用问题都可以使用。搜索算法的共同特征为:首先组成一组候选解;依据某些适应性条件测算这些候选解的适应度;根据适应度保留某些候选解,放弃其他候选解;对保留的候选解进行某些操作,生成新的候选解。

在遗传算法中,上述几个特征以一种特殊的方式组合在一起:基于染色体群的并行搜索,带有猜测性质的选择操作、交换操作和突变操作。这种特殊的组合方式将遗传算法与其他搜索算法区别开来。

遗传算法还具有以下几个优点:

(1)遗传算法是从问题解的串集开始搜索,而不是从某个解开始,这是遗传算法与传统优化算法的极大区别。传统优化算法是从单个初始值迭代求最优解的;容易误入局部最优解。遗传算法从串集开始搜索,覆盖面大,利于全局择优。

(2)许多传统搜索算法都是单点搜索算法,容易陷入局部的最优解。遗传算法同时处理群体中的多个个体,即对搜索空间中的多个解进行评估,减少了陷入局部最优解的风险,同时算法本身易于实现并行化,可以容易设计并行算法来加快搜索速度。

(3)遗传算法基本上不用搜索空间的知识或其他的辅助信息,而仅仅使用适应度函数值来评估单个个体,在此基础上进行遗传操作。适应度函数不仅不受连续可微的约束,而且其定义域可以任意设定。这一特点使得遗传算法的应用范围大大扩展。

(4)遗传算法不是采用确定性规则,而是采用概率的变迁规则来指导搜索方向。

(5)具有自组织、自适应和自学习性。遗传算法利用进化过程获得的信息自行组织搜索时,会把硬度大的个体赋值为具有较高的生存概率,并获得更适应环境的基因结构。

由于遗传算法所设计的整体搜索策略和优化搜索方法在计算时是不需要依赖于梯度信息或其他辅助知识,而只需要影响搜索方向的目标函数和相应的适应度函数,所以遗传算法提供了一种求解复杂系统问题的通用框架,它不依赖于问题的具体领域,所以广泛应用于许多科学。函数优化是遗传算法的经典应用领域,也是遗传算法进行性能评价的常用算例,许多人构造出了各种各样复杂形式的测试函数:连续函数和离散函数、凸函数和凹函数、低维函数和高维函数、单峰函数和多峰函数等。对于一些非线性、多模型、多目标的函数优化问题,用其他优化方法较难求解,而遗传算法可以方便的得到较好的结果。随着问题规模的增大,组合优化问题的

搜索空间也急剧增大，有时在目前的计算上用枚举法很难求出最优解。对这类复杂的问题，人们已经意识到应把主要精力放在寻求满意解上，而遗传算法是寻求这种满意解的最佳工具之一。实践已经证明，遗传算法对于组合优化中的NP问题非常有效。例如遗传算法已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面得到成功的应用。此外，遗传算法也在生产调度问题、自动控制、机器人学、图像处理、人工生命、遗传编码和机器学习等方面获得了广泛的运用。

模式识别(Pattern Recognition)也是人脑的一项基本智能。在日常生活中，人们无时无刻不在进行着“模式识别”工作。随着20世纪40年代计算机的出现以及50年代人工智能的兴起，人们当然也希望能用计算机来代替或扩展人类的部分脑力劳动。以计算机技术为基础的模式识别在20世纪60年代初迅速发展并成为一门新学科。

模式识别是指对表征事物或现象的各种形式的(包括数值的、文字的和逻辑关系的)信息进行处理和分析，以对事物或现象进行描述、辨认、分类和解释的过程，是信息科学和人工智能的重要组成部分。模式识别又常称作模式分类，从处理问题的性质和解决问题的方法等角度，模式识别分为有监督的分类和无监督的分类两种。二者的主要差别在于，各实验样本所属的类别是否预先已知。一般说来，有监督的分类往往需要提供大量已知类别的样本，比如说各种颜色的识别等，但在实际问题中，这是存在一定困难的，因此研究无监督的分类就变得十分有必要了。

模式还可分成抽象的和具体的两种形式，前者如意识、思想、议论等是属于概念识别研究的范畴，是人工智能的另一研究分支。我们所指的模式识别主要是对语音的波形、心电图、图片、照片、文字、符号、生物传感器等对象的具体模式进行辨识和分类。

模式识别研究主要集中在两方面，一是研究生物体是如何感知对象的，属于认识科学的范畴；二是在给定的任务下，如何用计算机实现模式识别的理论和方法。前者是生理学家、心理学家、生物学家和神经生理学家的研究内容，后者通过数学家、信息学专家和计算机科学工作者近几十年来的努力，已经取得了系统的研究成果。

应用计算机对一组事件或过程进行辨识和分类，所识别的事件或过程可以是文字、声音、图像等具体对象，也可以是状态、程度等抽象对象。这些对象与数字形式的信息相区别，称为模式信息。

模式识别所分类的类别数目由特定的识别问题决定。有时，开始时无法得知实际的类别数，需要识别系统反复观测被识别对象以后确定。

模式识别与统计学、心理学、语言学、计算机科学、生物学、控制论等都有关系。它与人工智能、图像处理的研究有交叉关系。例如自适应或自组织的模式识别系统包含了人工智能的学习机制；人工智能研究的景物理解、自然语言理解也包含模式识别问题。又如模式识别中的预处理和特征抽取环节应用图像处理的技术；图像处理中的图像分析也应用模式识别的技术。

模式识别方法主要有决策理论方法和句法方法两种。

决策理论方法又称统计方法，是发展较早也比较成熟的一种模式识别方法。首先将被识别对象进行数字化，变换为适于计算机处理的数字信息。一个模式常常要用很大的信息量来表示。许多模式识别系统在数字化环节之后还进行预处理，用于除去混入的干扰信息并减少某些变形和失真。随后进行特征抽取，即从数字化后或预处理后的输入模式中抽取一组特征。所谓

特征是选定的一种度量，它对于一般的变形和失真保持不变或几乎不变，并且只含尽可能少的冗余信息。特征抽取过程将输入模式从对象空间映射到特征空间。这时，模式可用特征空间中的一个点或一个特征矢量表示。这种映射不仅压缩了信息量，而且易于分类。在决策理论方法中，特征抽取占有重要的地位，但尚无通用的理论指导，只能通过分析具体识别对象决定选取何种特征。特征抽取后可进行分类，即从特征空间再映射到决策空间。为此而引入鉴别函数，由特征矢量计算出相应于各类别的鉴别函数值，通过鉴别函数值的比较进行分类。

句法方法也称结构方法或语言学方法。其基本思想是把一个模式描述为较简单的子模式（子句）的组合，子模式又可描述为更简单的子模式的组合，最终得到一个树形的结构描述，在底层的最简单的子模式称为模式基元。在句法方法中选取基元的问题相当于在决策理论方法中选取特征的问题。通常要求所选的基元能对模式提供一个紧凑的反映其结构关系的描述，又要易于用非句法方法加以抽取。显然，基元本身不应该含有重要的结构信息。模式以一组基元和它们的组合关系来描述，称为模式描述语句，这相当于在语言中，句子和短语用词组合，词用字符组合一样。基元组合成模式的规则，由所谓语法来指定。一旦基元被鉴别，识别过程可通过句法分析进行，即分析给定的模式语句是否符合指定的语法，满足某类语法的即被分入该类。

模式识别方法的选择取决于问题的性质。如果被识别的对象极为复杂，而且包含丰富的结构信息，一般采用句法方法；被识别对象不很复杂或不含明显的结构信息，一般采用决策理论方法。这两种方法不能截然分开，在句法方法中，基元本身就是用决策理论方法抽取的。在应用中，将这两种方法结合起来分别施加于不同的层次，常能收到较好的效果。

在信息技术及计算机技术日益普及的今天，如何将文字方便、快速地输入到计算机中已成为影响人机接口效率的一个重要瓶颈，也关系到计算机能否真正在人们的生活中得到普及和应用。目前，汉字输入主要分为人工键盘输入和机器自动识别输入两种。其中人工键入速度慢而且劳动强度大；自动识别输入又分为汉字识别输入及语音识别输入。从识别技术的难度来说，手写体识别的难度高于印刷体识别，而在手写体识别中，脱机手写体的难度又远远超过了联机手写体识别。到目前为止，除了脱机手写体数字的识别已有实际应用外，汉字等文字的脱机手写体识别还处在实验室阶段。

语音识别技术所涉及的领域包括信号处理、模式识别、概率论和信息论、发声机理和听觉机理、人工智能等等。近年来，在生物识别技术领域中，指纹识别技术以其独特的方便性、经济性和准确性等优势受到世人瞩目，并日益成为人们日常生活和工作中重要且普及的安全验证方式。目前语音识别方法在语音识别时识别速度较快，也有较高的识别率。

通过比较他的指纹和预先保存的指纹，便可以指纹识别，从而验证真实的身份。遥感图像识别已广泛用于农作物估产、资源勘察、气象预报和军事侦察等。在癌细胞检测、X 射线照片分析、血液化验、染色体分析、心电图诊断和脑电图诊断等方面，模式识别已取得了成效。

总的说来，人工智能研究的一个主要目标是使机器能够胜任一些通常需要人类智能才能完成的复杂工作。人工智能这门科学的具体目标也自然随着时代的变化而发展。它一方面不断获得新的进展，一方面又转向更有意义、更加困难的目标。目前能够用来研究人工智能的主要物质手段以及能够实现人工智能技术的机器就是计算机，人工智能的发展历史是和计算机科学与技术的发展史联系在一起的。人工智能目前在计算机领域内，得到了越来越广泛的重视，在机器人、经济政治决策、控制系统、仿真系统、游戏开发等领域中得到广泛应用。

2. 游戏中的人工智能

现代游戏的制作都是十分庞大的工程，目前还无法期望利用人工智能来创作整个的游戏。我们这里所说游戏的人工智能，是指用来控制游戏中各种活动对象行为的逻辑，使它们表现得合情合理，如同人的行为一样。游戏中活动对象分两类，一类是在背景中，如天上飘着的云，飞过的鸟。这类对象的行为连同它们的造型要显得逼真也不容易，需要掌握 2D 或 3D 的图形和动画技术，还需要有艺术修养。但它们在游戏中无须人工干预，变化不很多，控制的逻辑并不复杂。另一类活动对象是游戏中的各种角色，或称游戏代理，如虚拟的人、兽、怪物、机器人等。这些对象的活动方式必须变化多端才行，否则游戏就不好玩，所以控制逻辑就比较复杂。尤其是玩家对手的代理最困难。比如，要开发一个猫捉老鼠的游戏，假定玩家（不妨就是你）的代理是猫，则猫的行为由你操纵，而老鼠的行为则完全需要由程序来控制。当猫不出现时，老鼠必须到处觅食或打洞，以解决生存必需的食住问题，而一旦发觉有猫出现，则必须立即躲进洞里。如果附近没有洞，则要立刻逃窜，而逃窜的方向取决于它和猫的相对位置。如果猫在老鼠的西边，老鼠应向东逃跑；如果猫从老鼠东边追来，则老鼠应向西跑；如果途中遇到障碍物挡住了去路，则应改变方向，如向南或向北跑，至少不应该回头跑，除非前面是死胡同。老鼠能遵循这样的逻辑来行动，就是游戏编程中为老鼠设计的智能，是游戏人工智能。同样，猫也需要人工智能，但很简单，那就是听话，能听从你用按键或鼠标进行的指挥。

所有角色扮演类游戏都需要有类似的智能。愈是好玩的游戏需要的智能愈复杂。但并不是所有游戏都要有人工智能，例如 Windows 提供的接龙和挖地雷游戏就没有人工智能问题；网上供两人对弈的象棋、围棋、军棋类游戏也不需要人工智能。如果要由机器当公证人，那也只需要很低级的智能。但一旦要求计算机能与人对弈，那就需要一定的智能了。

策略类人工智能可以说是 AI 中比较复杂的一种，最常见的策略类 AI 游戏就是棋盘式游戏。在这类游戏中，通常的策略类 AI 程序都是使计算机判断目前状况下所有可走的棋与可能的获胜状况，并计算当前计算机可走棋步的获胜分数或者玩家可走棋步的获胜分数，然后再决定出一个最佳走法。

人工智能在计算机上实现有两种不同的方式。一种是采用传统的编程技术，使系统呈现智能的效果，而不考虑所用方法是否与人或动物机体所用的方法相同。这种方法称为工程学方法，它已在一些领域内作出了成果，如文字识别、电脑下棋等。另一种是模拟法，它不仅要看效果，还要求实现方法也和人类或生物机体所用的方法相同或相类似。遗传算法模拟人类或生物的遗传——进化机制，人工神经网络则是模拟人类或动物大脑中神经细胞的活动方式。为了得到相同智能效果，两种方式通常都可使用。采用前一种方法，需要人工详细规定程序逻辑，如果游戏简单，还是方便的。如果游戏复杂，角色数量和活动空间增加，相应的逻辑就会很复杂（按指数式增长），人工编程就非常繁琐，容易出错。而一旦出错，就必须修改源程序，重新编译、调试，最后为用户提供一个新的版本或提供一个新补丁，非常麻烦。采用后一种方法时，编程者要为每一个角色设置一个智能系统（一个模块）来进行控制，这个智能系统（模块）开始什么也不懂，就像初生婴儿那样，但它能够学习，能渐渐地适应环境，应付各种复杂情况。这种系统开始也常犯错误，但它能吸取教训，下一次运行时就可能改正，至少不会永远错下去，不必发布新版本或打补丁。利用这种方法来实现人工智能，要求编程者具有生物学的思考方法，入门难度大一点。由于模拟法编程时无须对角色的活动规律作详细规定，应用于复杂问题，通常会比前一

种方法更省力。

将要解决的问题转换到对应计算机的状态空间中，然后以状态空间的分析方法分析问题，是解决简单问题常用的有效方法。在一个比较大的智能游戏系统中，局部的智能表象模拟往往都是采用这种简单而行之有效的方法，因为计算速度可以很快。举例来说：有四个人晚上过桥，但只有一个手电筒，使用手电筒可以同时两个人过桥。单独过桥需要时间分别为 10、5、2、1 分钟，如果有两个人同时过桥将按慢的人算时间，问最少多长时间这些人都能过去桥？

我们可以先建立状态表示模型，然后状态转化模型（方便计算机进行处理），最后智能评估模型。状态表示模型即能表示某一时刻的状态。状态通常由一些状态参数或规则表示。建立模型 A：

Time：已用时间。

PersonState[4]：4 个人的状态，可以用“0”表示未过桥，“1”表示已经过桥。

PersonWalkTime[4]：4 个人的走路时间。

LightState：手电筒状态，用“0”表示未过桥，“1”表示已过桥。

状态转化模型：由一状态如何转化到下一状态。比如模型 B：加最简单限制，if(LightState==0)可两人过桥，否则只许一人过桥，并且所有状态发生相应变化。采用广度搜索算法进行所有可能的状态转化。

模型 C：时间（Time）越小越优。

在这个系统建立完毕之后，很容易使用广度搜索算法来找到最佳过河方案。若是把相应数据输入电脑，而电脑可以得出过桥方式，那么就可认为在一定程度这个系统具有了智能的表现。一般来说模型定义的越好，系统的性能也越好。

可加入一些改进措施使我们的系统更加的“智能”一些。我们尽量使走得慢的人过桥次数少一些，因此我们在每次返回一个携带手电筒的人时，都选择走的快的，这样的话，我们的搜索就显的更智能了一些，速度也会快很多了。可以在开始阶段定义一个 Time 数值，若出现搜索过程中 Time 已超过此数值的，其后面搜索不再进行。若出现搜索结果小于 Time 的，我们更新 Time，并继续搜索。运用这种方法，我们将省去很多无用的搜索。

棋类游戏就是一种智力游戏，被研究的也比较深入。黑白棋，又叫反转棋、奥赛罗棋等，在西方很流行，它的规则比较简单，上手容易。黑白棋的棋盘是一个有 8×8 方格的棋盘。下棋时将棋下在空格中间，而不是像中国象棋一样下在交叉点上。开始时在棋盘正中有一白一黑四个棋子交叉放置，黑棋是先下子方。下子的方法是：把自己颜色的棋子放在棋盘的空格上，而当自己放下的棋子在横、竖、斜八个方向内有一个自己的棋子，则被夹在中间的全部会成为自己的棋子。并且，只有在可以翻转棋子的地方才可以下子。最后棋盘全部占满，子多者为胜。

战略点权值表：棋盘的四个角是永远不能被吃掉的，所以这四个点是战略要地。同样，我们可以得出另一些重要的战略点。我们把这些点赋一个较大的权重，这样，我们可得到一个权重表（对应棋盘）。当然这个表可使用一些参数，并且可随过程改变而改变，越灵活的格式将产生越好的智能表象。局势评估模块：对敌我双方的总体或局部兵力分布得出一个较为合理的分析结果，有了这个模块，才能更好地把握战局。

上述算法虽然可以模拟出智能表象，但遗憾的是没有学习功能。而学习功能相对于智能生物是一个非常重要的功能。因此我们可以使用遗传算法来进行改进。棋盘大小为 64 格，每个

格状态采用－1 代表敌人的棋子、0 代表未下子、1 代表自己的棋子。对每一个格子用 5×5 的局部棋子来评价其战略重要性。这样我们就得到一个 W[64][25]的表，对于每一种棋盘局势它总能判断出战略重要性最大的点。这种编码方式考虑了棋盘全局位置特性与局部局势特性。选取种群大小为 30，随机生成 30 个 Wi[64][25]。

初始训练阶段：对每一个 Wi[64][25]，让其判断一些特定的战略点，判断力好的适应度大。后期训练阶段：和人对下，赢的多的适应度大。选择适应度大的 10 个为产生优良种群 goodX(t)。对 goodX(t)应用遗传算子，产生新一代群体 X(t＋1)(30 个个体)。交叉遗传，可选取 father[0－32][25]和 mother 的[32－64][25]产生新的个体。变异遗传，对 father[64][25]中的数值产生一些随机变化从而生成新个体。t＝t＋1；如果不满足终止条件继续上面操作。采用这种方式，我们可以使程序自己学习，从而模拟智能。遗传算法的主体就是对问题编码，然后通过进化方式进化出优秀(相对于适应度)的种群，编码方式由具体问题决定，进化方式影响进化范围和速度。通过这种方式的学习，模型一般都能得到此编码方式下的最优编码，从而具有超越人为编码的性能。有兴趣的读者可以自行深入研究。

游戏中人工智能设计是非常关键的，是吸引游戏玩家的重要方法，毕竟我们不喜欢与“傻乎乎”的敌人进行对战，不费力气的获得胜利。

3. 五子棋游戏中人工智能的设计

要使计算机可以“智能的”下子实际上是分两个基本步骤的，第一个步骤是尽可能的收集棋盘格局的信息，这对将来进行分析至关重要，并且要把这些信息以一定的格式存放在内存中。第二步，对收集到的信息进行分析处理，即要给出一个规则，用穷举搜索的办法遍历所有收集到的信息，搜索的过程实际上是用所定下的规则去衡量每一点的权值，可以用另外的一个整数数组来保存这些权值。搜索权值数组的目的是为了找到一个权值最大的点，这个点就是当前的最优点，也就是应该下子的位置。

具体说来，在第一步中，可以用一个 15×15 的二维数组来存放棋盘上每一个点的信息。一般可以考虑是一个二维的整数数组，某元素如果值为 1 代表此点是黑子，0 代表是白子，2 代表此处还没有下子。每下一步棋，就用一个 15×15 的二重循环去遍历棋盘上的每一个点，可以参照如下的方法，即对于每一个点，我们假定这个点放上黑子，这时候就判断这个黑子放上去后，会形成几个子相连的情况，然后把对应的数值填入上面所说的二维数组里面，然后再假定这个点放白棋，又会形成多少个子相连的情况，也填入二维数组里面。当然也可以用两个二维数组分别存储黑子和白子的情况，这样遍历完棋盘后，数组里面就保存了有用的棋盘格局信息。

一个比较完整的五子棋的 AI 构想可以考虑如下信息：

(1)预先设定好棋盘上每一种情况的权值；

(2)计算棋盘中每一个可落子点的权值(分数；)

(3)计算机要具有一定的攻击和防守能力；

(4)权值最大的落子点就是下子的最好点。

下面对上述信息进行详细分析：

(1)预先设定棋盘上各种情况的权值。

权值的设定就是 AI 的核心思想，也就是模拟人的思维的核心，需要仔细的考虑各种情况，还需要考虑各种情况应该给多少权值。

a. 如果要落子的点正好是计算机取胜点，即五子相连的点，那么考虑把此点的权值设定为我们这里的最大值 100000，即告诉计算机一定要在此处下子，如果有很多个可以五子相连的点也没有关系，因为只要随意下一个即可以取胜；

b. 如果要落子的点正好是对手的取胜点，那么考虑把此点的权值设定为一个非常大的值 50000。这个值的设计不能大于第一种情况的权值，因为我们的目的就是要取胜，既然已经可以取胜那么自然不需要防守；

c. 如果要落子的点是会让计算机的四个子相连的点，那么这个点是一个比较好的落子点，但是还没有前两种情况的特别需要考虑，所以考虑把此点的权值设定为一个比较大的值 10000；

d. 如果要落子的点是会让对手的四个子（即现在对手已经三子相连）相连的点，那么这个点是对手一个比较好的点，对计算机自己当然不是很有利的点了，考虑把此点的权值设定为 5000；

e. 如果要落子的点是会让计算机的三个子相连的点，理由和前边第三种情况一样，但是没有第三种和第四情况重要，所以此点权值不能高于第三或第四种情况。考虑把此点的权值设定为 1000；

f. 如果要落子的点是会让对手的三个子相连的点，那么考虑把此点的权值设定为 500，理由同前；

g. 如果要落子的点是会让计算机的两个子相连的点，那么考虑把此点的权值设定为 100；

h. 如果要落子的点是会让对方的两个子相连的点，那么考虑把此点的权值设定为 50。

如果某点同时满足了上述的两条或多条，那么可以考虑把同时满足的情况的权值的和作为最终这点的权值，这样就等于描述了如果下在某一个点对我方（计算机方）会有多重要的好处。

基于以上这种权值的设定使得计算机有了一定的思考，即通过权值高低来区分重要和不重要的情况，但是在这种设计下计算机对于棋盘全局没有特别的考虑，不能区分如果两点具有同样权值的时候该在哪里下子，读者可以自行设计一个加强算法来解决这个问题。

(2)计算棋盘上每一个点的权值。

可以使用二维循环来遍历棋子数组，分别计算和设定每一个地方的权值。如果在某一点处已经下了子，那么把这个点的权值直接赋值为－1，这样就代表了此处不可以下子，因为其他点总是会有正的权值出现，那样就一定会大于此处的－1；如果此处还没有子，则可以假定这个地方如果下我方棋子的情况，然后按照上边的各种情况的权值取值对这个点进行权值指定。

(3)计算机的攻击和防守。

要使计算机具有防守和攻击两种能力，那么就不能在权值设定时只假设落计算机方的棋子，还要考虑这个点如果下对手的棋子会出现的情况。如果对手最佳攻击位置的权值大于计算机最佳攻击位置上的权值，那么计算机就将下一步的棋子摆在对手的最佳攻击位上以阻止对手的进攻，否则计算机便将棋子下在自己的最佳攻击位置上进行攻击。例如，计算机方在某处落子只是会出现计算机方两子相连，但是如果对手在这个地方下棋子会出现对手四子相连的情况，那么就应该把这个点权值再提高。在计算每一个可以下子点的权值的时候，最后计算得到的权值应该是计算机方在此点下棋子的权值和对手在此点下棋子的权值之和。

(4)最佳落子点。

当棋盘上每一个点的权值都已经赋值完以后,可以使用二维循环来遍历权值数组,选择权值最大的点,此点就是计算机落子的最佳点,然后安排计算机在此处下子。这样计算机就完成了一次思考,选择了某一点进行下子。

2.3 游戏实施

2.3.1 基本实施

在游戏的基本实施中,主要是模拟无人工智能参与的游戏过程,故实现过程简单,减少了计算机(Computer)类。

1. 棋盘类

棋盘类 ChessBoard 代码如下:

ChessBoard.cs

```
publicclass ChessBoard
{
    //棋子对象
    private Stones stone=new Stones();
    //arrchessboard 为棋盘情况数组,arrchessboard[i,j]=1 表示此处为黑子,
    //arrchessboard[i,j]=2 表示此处为白子
    private int[,] arrchessboard=new int[16, 16];

    //绘制棋盘
    public void DrawBoard(Graphics g)
    {
        Point point;
        for (int y=50; y < 541; y += 35)
        {
            for (int x=50; x < 541; x += 35)
            {
                point=new Point(x, y);
                Pen p=new Pen(Color.Black);
                g.DrawRectangle(p, point.X, point.Y, 35, 35);
            }
        }

    }
    //下棋并判断结果
    public void DownStone(int m, int n, int count, Graphics g)
```

```
{
    stone.X=(float)(m * 35+50-17.5);
    stone.Y=(float)(n * 35+50-17.5);

    //先画出黑棋(黑白规则不一致)
    if (count % 2 == 1)
    {
        if (arrchessboard[m, n] ! = 1 && arrchessboard[m, n] ! = 2)
        {
            stone.DrawBlackStone(stone.X, stone.Y, g);
            arrchessboard[m, n]=1;
            if (Rule.Result(m, n, arrchessboard) > 0)
            {
                MessageBox.Show("黑棋胜利!");
                Form1 f=new Form1();
                //f.Refresh();
                f.Show();
            }
        }
    }
    if (count % 2 == 0)
    {
        if (arrchessboard[m, n] ! = 1 && arrchessboard[m, n] ! = 2)
        {
            stone.DrawWhiteStone(stone.X, stone.Y, g);
            arrchessboard[m, n]=2;
            if (Rule.Result(m, n, arrchessboard) > 0)
            {
                MessageBox.Show("白棋胜利!");
                Form1 f=new Form1();
                f.Show();
            }
        }
    }
}
```

2. 棋子类

棋子类代码如下：

Stones.cs

```
publicclass Stones
{
    //棋子坐标
```

```
    private float x;
    private float y;

    public float X
    {
        get { return x; }
        set
        {
            x=value;
        }
    }
    public float Y
    {
        get { return y; }
        set
        {
            y=value;
        }
    }
    //绘制棋子
    public void DrawBlackStone(float x_stone, float y_stone, Graphics g)
    {
        x=x_stone;
        y=y_stone;
        SolidBrush bB=new SolidBrush(Color.Black);
        g.FillEllipse(bB, x, y, 30, 30);
    }
    public void DrawWhiteStone(float x_stone, float y_stone, Graphics g)
    {
        x=x_stone;
        y=y_stone;
        SolidBrush b=new SolidBrush(Color.White);
        g.FillEllipse(b, x, y, 30, 30);
    }
}
```

3. 规则类

规则类 Rule 代码如下：

Rule.cs

```
publicclass Rule
{
```

```
public static int Result(int m, int n, int[,] arrchessboard)
{
    //m,n 点四个方向的连子数,依次正东正西,正南正北方,西北东南,西南东北
    int[] arrf=new int[4];
    arrf[0]=Xnum(m, n, arrchessboard);
    arrf[1]=Ynum(m, n, arrchessboard);
    arrf[2]=YXnum(m, n, arrchessboard);
    arrf[3]=XYnum(m, n, arrchessboard);
    for (int i=0; i < 4; i++)
    {
        //检查四个方向,看是否有连成个
        if (Math.Abs(arrf[i]) == 5)
        {
            return 1;
        }
    }
    return 0;
}
public static int Xnum(int m, int n, int[,] arrchessboard)
{

    //连子个数
    int num=1;
    //正东方向检查(x+)
    int i=m+1;
    //不超出棋格
    while (i < 15)
    {
        //前方的棋子与 m,n 点棋色不同时跳出循环
        if (arrchessboard[i, n] == arrchessboard[m, n])
        {
            num++;
            i++;
        }
        else
        {
            break;
        }
    }

    //正西方向检查(x-)
    i=m - 1;
```

```
        while (i >= 0)
        {
            //前方的棋子与 m,n 点不同时跳出循环
            if (arrchessboard[i, n] == arrchessboard[m, n])
            {
                num++;
                i--;
            }
            else
            {
                break;
            }
        }
         return num;
    }
    /// <summary>
    ///正南正北方向检测
    ///<returns>如果返回负值则表示改方向在无子可下</returns>
    public static int Ynum(int m, int n, int[,] arrchessboard)
    {
        //连子个数
        int num=1;
        //正南方向检查(y+)
        int i=n+1;
        while (i < 15)
        {
            //前方的棋子与 m,n 点不同时跳出循环
            if (arrchessboard[m, i] == arrchessboard[m, n])
            {
                num++;
                i++;
            }
            else
            {
                break;
            }
        }
       //正北方向检查(y-)
        i=n - 1;
        while (i >= 0)
        {
            //前方的棋子与 m,n 点不同时跳出循环
```

```
            if (arrchessboard[m, i] == arrchessboard[m, n])
            {
                num++;
                i--;
            }
            else
            {
                break;
            }
        }
        return num;
    }
    /// <summary>
    ///西北东南方向检查
    /// <returns>如果返回负值则表示改方向在无子可下</returns>
    public static int YXnum(int m, int n, int[,] arrchessboard)
    {
        //连子个数
        int num=1;
        //东南方向(x+,y+)
        int i=m+1;
        int j=n+1;
        //不超出棋格
        while (i < 15 && j < 15)
        {
            //前方的棋子与m,n点不同时跳出循环
            if (arrchessboard[i, j] == arrchessboard[m, n])
            {
                num++;
                i++;
                j++;
            }
            else
            {
                break;
            }
        }
        //西北方向(x-,y-)
        i=m - 1;
        j=n - 1;
        //不超出棋格
        while (i >= 0 && j >= 0)
```

```
        {
            //前方的棋子与 m,n 点不同时跳出循环
            if (arrchessboard[i, j] == arrchessboard[m, n])
            {
                num++;
                i--;
                j--;
            }
            else
            {
                break;
            }
        }
        return num;
    }
    /// <summary>
    ///西南东北方向检查
    ///<returns>如果返回负值则表示改方向在无子可下</returns>
    public static int XYnum(int m, int n, int[,] arrchessboard)
    {
        //连子个数
        int num=1;
        //西南方向(x-,y+)
        int i=m - 1;
        int j=n+1;
        //不超出棋格
        while (i >= 0 && j < 15)
        {
            //前方的棋子与 m,n 点不同时跳出循环
            if (arrchessboard[i, j] == arrchessboard[m, n])
            {
                num++;
                i--;
                j++;
            }
            else
            {
                break;
            }
        }
        //东北方向(x+,y-)
        i=m+1;
```

```
            j=n - 1;
            //不超出棋格
            while (i < 15 && j >= 0)
            {
                //前方的棋子与 m,n 点不同时跳出循环
                if (arrchessboard[i, j] == arrchessboard[m, n])
                {
                    num++;
                    i++;
                    j--;
                }
                else
                {
                    break;
                }
            }
            return num;
        }
    }
```

4. 窗体实现

窗体的实现代码如下：

Form1.cs

```
public partialclass Form1 : Form
    {
        private bool start=false;
        private int count=1;//记录下棋次数
        private Stones stone=new Stones();//棋子对象
        private ChessBoard chess=new ChessBoard(); //棋盘对象
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            //画棋盘
            Graphics g=e.Graphics;
            chess.DrawBoard(g);
        }
        private void button_Start_Click(object sender, EventArgs e)
        {
            //开始游戏
            start=true;
            button_Start.Enabled=false;
        }
```

```
        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {

            if (start && e.Button == MouseButtons.Left)
            {
                if (e.X < 576 && e.Y < 576 && e.X > 50 && e.Y>50)
                {
                    Graphics g=this.CreateGraphics();

                    //取下棋点(坐标必须落在棋盘交点处)
                   // int m=(int)Math.Floor((decimal)(e.X-50)/35);
                    int m=(int)((e.X - 50) / 35);
                    int n=(int)((e.Y - 50) / 35);

                    chess.DownStone(m,n,count,g);
                    g.Dispose();
                    count++;
                }
            }
        }
    }
```

2.3.2 高级实施——人工智能实现

根据设计思路,计算机落棋位置的选择是通过计算各个落子点权值大小的方法实现的。在计算出每个可落子点的权值之后,还需要一个从所有权值中找出最大值的辅助方法。

在项目中添加一个 Computer 类,具体实现代码如下:

Computer.cs

```
///电脑类。计算电脑下棋的位置
/// </summary>
publicclass Computer
{
private bool mflag;//表明计算机是下黑棋还是下白棋,true 是黑棋,false 是白棋
private int x;//计算机下子的位置 X 坐标
private int y;//计算机下子的位置 X 坐标

public Computer(bool flag)
{
    mflag=flag;
}
    //定义属性
```

```
public int X
{
    get
{
        return x;
    }
}
    //定义属性
public int Y
{
    get
    {
        return y;
    }
}

/// <summary>
///计算机下棋
/// </summary>
/// <param name="arrchessboard"></param>
public void Down(int[,] arrchessboard)
{
    //定义权值数组,记录下来棋盘上每一个点的权值
    int [,] qz=new int[15,15];

        //二重循环遍历,计算每个点的权值
        for (int i=0;i<15;i++)
        {
            for(int j=0;j<15;j++)
            {
            if (arrchessboard[i,j] < 2)
            {
                //当已有子时标注权值为-1,表示不可以下子
                qz[i,j]=-1;
            }
            else
            {
                //计算这个点权值
            qz[i,j]=Check(i,j,arrchessboard);
                    }
                }
            }
```

```
        //找出权值最大的点,就是计算机下子的最好点
        FindMaxQZ(qz);
    }

    /// <summary>
    ///找出权值最大点
    /// </summary>
    /// <param name="qz"></param>
    private void FindMaxQZ(int [,] qz)
    {
        int max=0;
        for (int i=0;i<15;i++)
        {
            for(int j=0;j<15;j++)
            {
                if (qz[i,j]>max)
                {
                    //记录下来权值最大点出的(x,y)坐标
                    x=i;
                    y=j;
                    //记录下来最大权值
                    max=qz[i, j];
                }
            }
        }
    }

    /// <summary>
    ///设定棋盘上(m,n)点的权值
    /// </summary>
    /// <param name="m"></param>
    /// <param name="n"></param>
    /// <param name="arrchessboard"></param>
    /// <returns></returns>
    private int Check(int m, int n, int[,] arrchessboard)
    {
        int qz=0;

        //找计算机自己的取胜点,即子相连的点
        int w1=100000;
        //找对手的取胜点
        int w2=50000;
```

```
//找自己的三个相连的点
int w3=10000;
//找对手的三个相连的点
int w4=5000;

//找自己的两个相连的点
int w5=1000;
//找对手的两个相连的点
int w6=500;

//找自己的相连的点
int w7=100;
//找对方的相连的点
int w8=50;

//找自己的失败点
int w9=-1000000;

int[] arrf=new int[4];

//如果在该位置下计算机方的子
if (mflag)
{
    //计算机是黑子
    arrchessboard[m,n]=0;
}
else
{
    //计算机是白子
    arrchessboard[m,n]=1;
}

//检查个方向的连子情况
arrf[0]=Rule.Xnum(m,n,arrchessboard);
arrf[1]=Rule.Ynum(m,n,arrchessboard);
arrf[2]=Rule.YXnum(m,n,arrchessboard);
arrf[3]=Rule.XYnum(m,n,arrchessboard);

//中心点权值加
if (m==7 && n==7)
{
    qz+=1;
```

```
}

for (int i=0;i<4;i++)
{
    if (Math.Abs(arrf[i]) == 5)
    {
        //如果在此点下子计算机方子相连,加相应权值
        qz += w1;
    }
    if (arrf[i] == 4)
    {
        //如果在此点下子计算机方子相连,加相应权值
        qz += w3;
    }
    if (arrf[i] == 3)
    {
        //如果在此点下子计算机方子相连,加相应权值
        qz += w5;
    }
    if (arrf[i] == 2)
    {
        //如果在此点下子计算机方子相连,加相应权值
        qz += w7;
    }

    //如果计算机方为黑棋,还要检查是不是不允许下子的点,如双三禁手等
    if (mflag)
    {
        if (Rule.IsFail(arrf, arrchessboard[m,n])>0)
        {
            qz += w9;
        }
    }
}

//如果该位置下对方的子
if (mflag)
{
    //对方是白子
    arrchessboard[m,n]=1;
}
else
```

```
        {
            //对方是黑子
            arrchessboard[m,n]=0;
        }

        //检查每个方向的连子情况
        arrf[0]=Rule.Xnum(m,n,arrchessboard);
        arrf[1]=Rule.Ynum(m,n,arrchessboard);
        arrf[2]=Rule.YXnum(m,n,arrchessboard);
        arrf[3]=Rule.XYnum(m,n,arrchessboard);
        for (int i=0;i<4;i++)
        {
            if (Math.Abs(arrf[i]) == 5)
            {
                qz += w2;
            }
            if (arrf[i] == 4)
            {
                qz += w4;
            }
            if (arrf[i] == 3)
            {
                qz += w6;
            }
            if (arrf[i] == 2)
            {
                qz += w8;
            }
        }
        //检查完后把(m,n)点恢复到无子时的默认值
        arrchessboard[m,n]=2;
        return qz;
    }
}
```

在 Chessboard 类中添加 ComputerDownStone()函数，代码如下：

```
/// <summary>
///电脑下棋
/// </summary>
private void ComputerDownStone()
{
```

```
int m,n,count=0;
do
{
    computer.Down(arrchessboard);
    m=computer.X;
    n=computer.Y;
    count++;
    if (count > 100)
    {
        MessageBox.Show("异常!");
        Start();
        return;
    }
}
while(Rule.IsExist(m,n,arrchessboard));

DownStone(m,n);
}
```

修改 Chessboard 类中 PersonDownStone 方法，在人下完棋子以后就进行计算机下棋子，代码如下：

```
/// <summary>
///人下棋
/// </summary>
/// <param name="x"></param>
/// <param name="y"></param>
public void PersonDownStone(int x,int y)
{
    if (x < 600 && y < 600)
    {
        //取下棋点
        int m=(int)Math.Floor(x/40.0);
        int n=(int)Math.Floor(y/40.0);
        if (! Rule.IsExist(m,n,arrchessboard))
        {
            DownStone(m,n);
            ComputerDownStone();
        }
    }
}
```

修改 Chessboard 类中 Start 方法，如果计算机是先手的话就调用 ComputerDownStone()